DOCUMENTO DI ARCHITETTURA

Il documento include diagrammi delle classi e codice in OCL

Andrea Gravili, Maria Laura La Face, Matteo Parma

INDICE

INDICE..... 2
SCOPO DEL DOCUMENTO..... 3
DIAGRAMMA DELLE CLASSI..... 3
UTENTI..... 4
LOGIN E AUTENTICAZIONE..... 5
PAGINA HOME..... 6
GESTIONE UTENTI..... 7
GESTIONE TASK..... 8
DIAGRAMMA DELLE CLASSI COMPLETO..... 9
CODICE IN OBJECT CONSTRAINT LANGUAGE..... 10
AUTORIZZAZIONE LOGIN..... 11
UTENTE LOGGATO..... 12
GESTIONE delle TASK..... 12
DIAGRAMMA DELLE CLASSI E CODICE IN OCL..... 14

SCOPO DEL DOCUMENTO

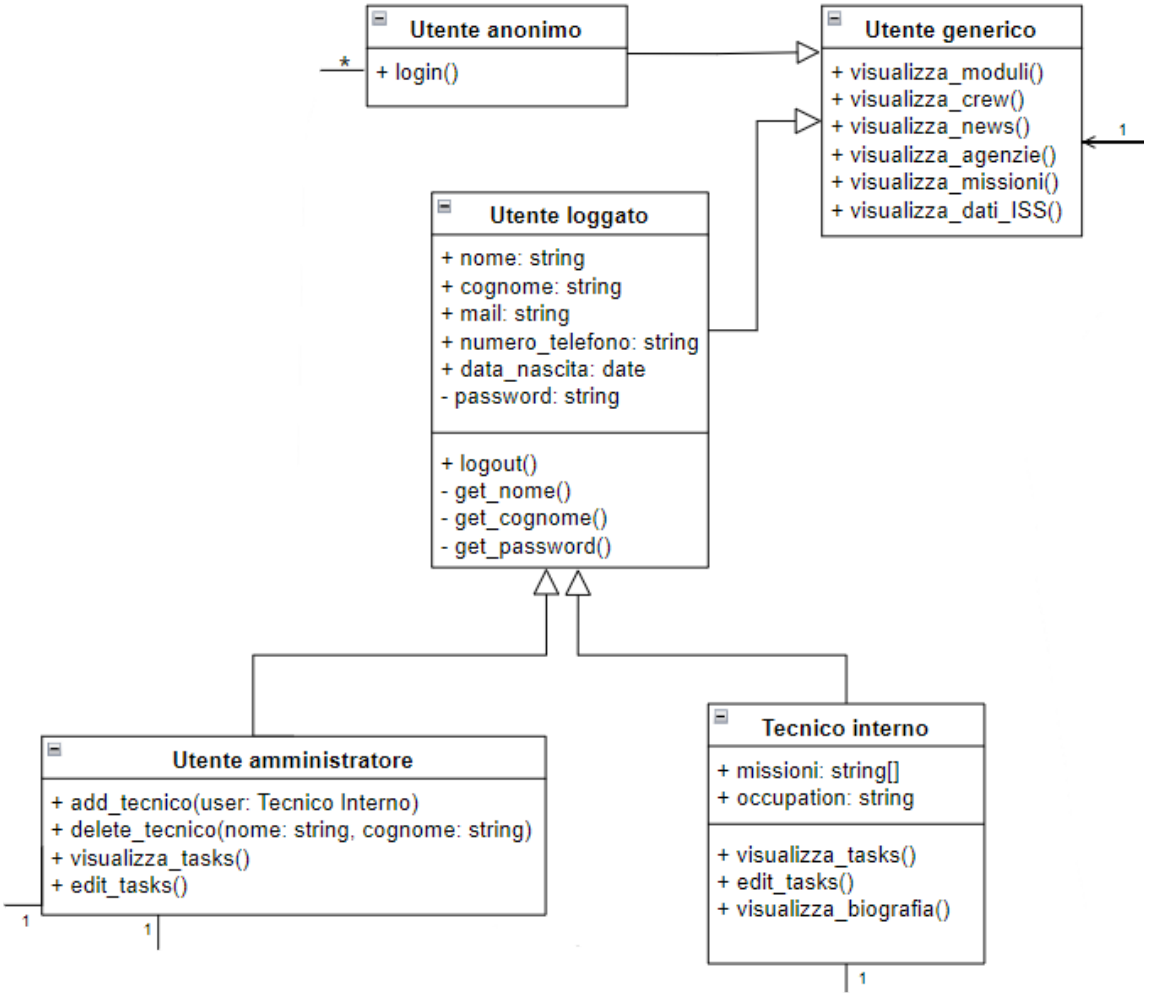
Il documento in oggetto riporta la definizione dell'architettura del progetto ISS4U utilizzando il diagramma delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento D3 sono stati descritti il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definita l'architettura del sistema, dettagliando da un lato le classi che dovranno essere implementate a livello di codice, e dall'altro la logica che regola il comportamento del software. Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML, mentre la logica viene descritta in OCL.

DIAGRAMMA DELLE CLASSI

In questo capitolo vengono riportate le classi previste all'interno del progetto ISS4U. Le classi sono state individuate tramite lo studio dei requisiti funzionali del D1 e dei componenti del diagramma dei componenti del D2. Ogni classe viene caratterizzata da un nome, una lista di attributi (che identificano i dati gestiti dalla classe) e una lista di metodi (che descrivono tutte le operazioni previste all'interno della classe). Le classi possono essere associate ad altre classi, per questo motivo all'interno del diagramma saranno presenti cardinalità e altre informazioni su come le classi interagiscono tra loro.

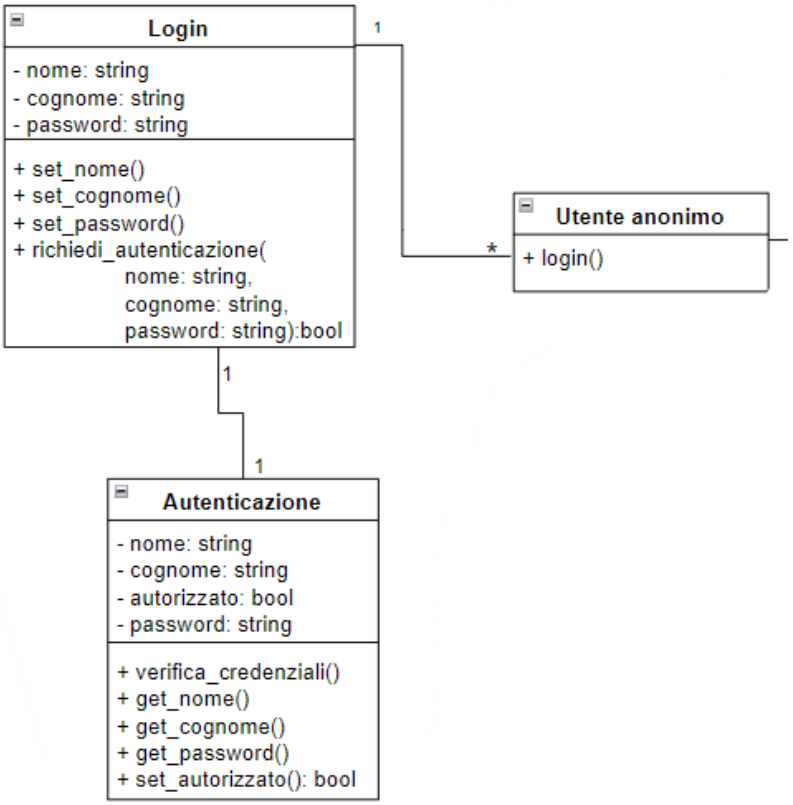
UTENTI

Analizzando i requisiti funzionali e il diagramma dei componenti, si possono individuare diverse tipologie di utente: l'**Utente generico**, che e' colui che accede a funzionalità ristrette del nostro sistema ISS4U (quali la navigazione della pagina home e la visualizzazione delle varie categorie), l'**Utente anonimo**, che ha la possibilità di effettuare il login, l'**Utente loggato**, che può fare le stesse cose dell'utente generico ma viene registrato all'interno del database tramite il suo account; quest'ultimo può essere di due tipi, il **Tecnico interno** e l'**Utente amministratore**, i quali condividono gli stessi attributi dell'utente loggato ma possono svolgere attività diverse (ex. rispettivamente la visualizzazione delle task e l'aggiunta di tecnici interni).



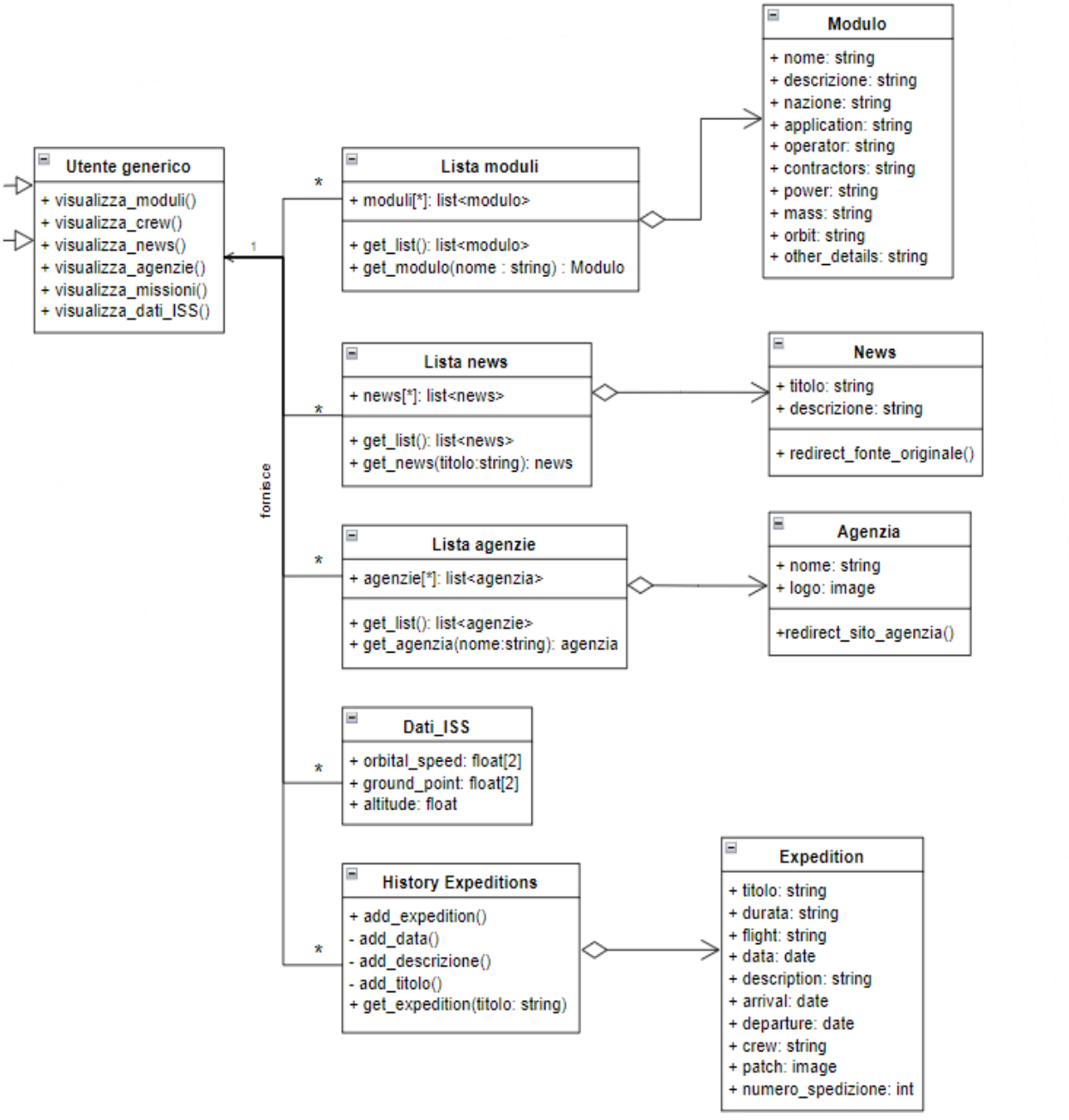
LOGIN E AUTENTICAZIONE

L'utente anonimo ha la possibilità di effettuare il login al sistema. Dovrà quindi compilare un form di login che prende in input il nome e cognome e la password, che dovranno essere verificati nel processo di autenticazione. Per questo motivo, dopo aver analizzato il diagramma dei componenti, sono state identificate la classe di **Login** e di **Autenticazione** che si occupano appunto della gestione delle credenziali.



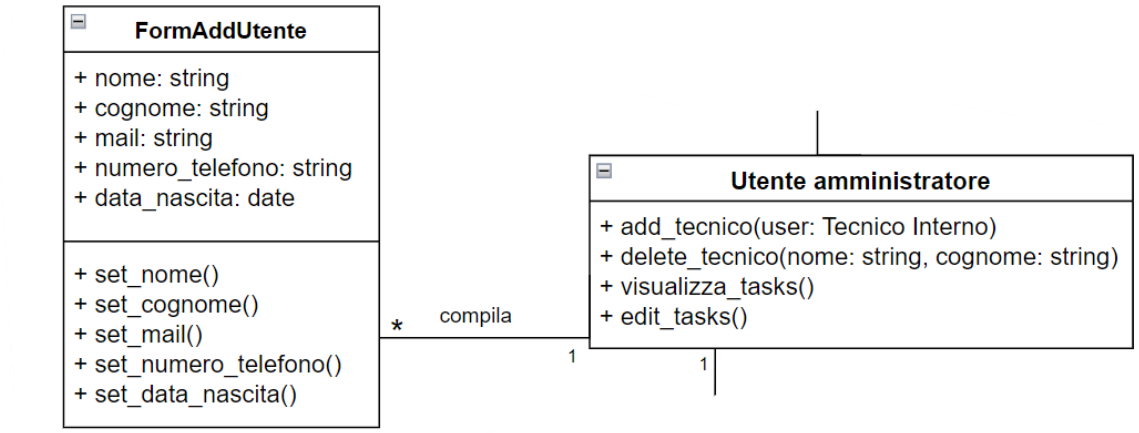
PAGINA HOME

Analizzando il diagramma dei componenti, sono state identificate le seguenti classi: **Moduli**, **News**, **Agenzie**, **Dati ISS** e **Expeditions**. Ognuna di queste classi corrispondono alle diverse pagine che l'utente generico può visualizzare all'interno della Home. Queste classi, ad eccezione di Dati ISS, sono in realtà delle aggregazioni delle seguenti classi, rispettivamente **Lista moduli**, **Lista news**, **Lista agenzie** e **History Expeditions**. Questo per indicare come vengono presi i dati dal database per ogni categoria.



GESTIONE UTENTI

Come descritto nel requisito funzionale 10, una delle funzionalità dell'utente amministratore è quella di poter aggiungere e/o rimuovere utenti. È stata quindi individuata una classe **FormAddUtente**, in quanto il processo di aggiunta di un utente avviene tramite un form, dentro al quale bisogna indicare tutti i dati necessari, come descritto negli attributi e nei metodi di questa classe.



GESTIONE TASK

Analizzando il diagramma dei componenti e i requisiti funzionali, sono state identificate le seguente classi che si occupano di gestire le task: **FormSelectUser**, che l'Utente amministratore compila quando deve creare una task per un tecnico interno, **Lista task**, **FormAddTask**, **FormModificaTask**, dove *Lista Task* permette di passare la lista delle task alla pagina, *FormAddTask* ne permette l'aggiunta e *FormModificaTask* permette la modifica di una o più task, e infine la classe **Task**, che tramite i suoi attributi descrive il formato di tutte le task.

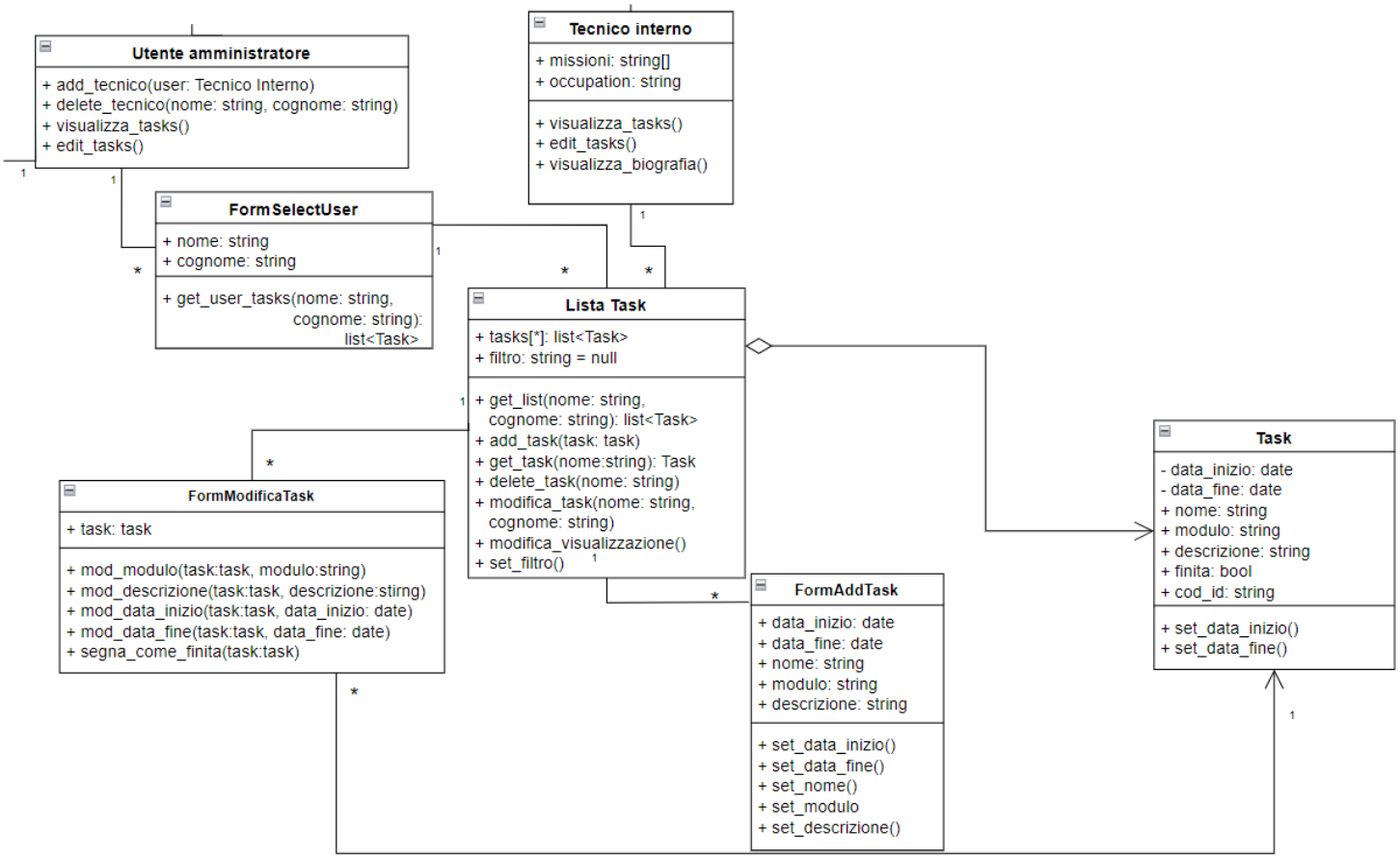
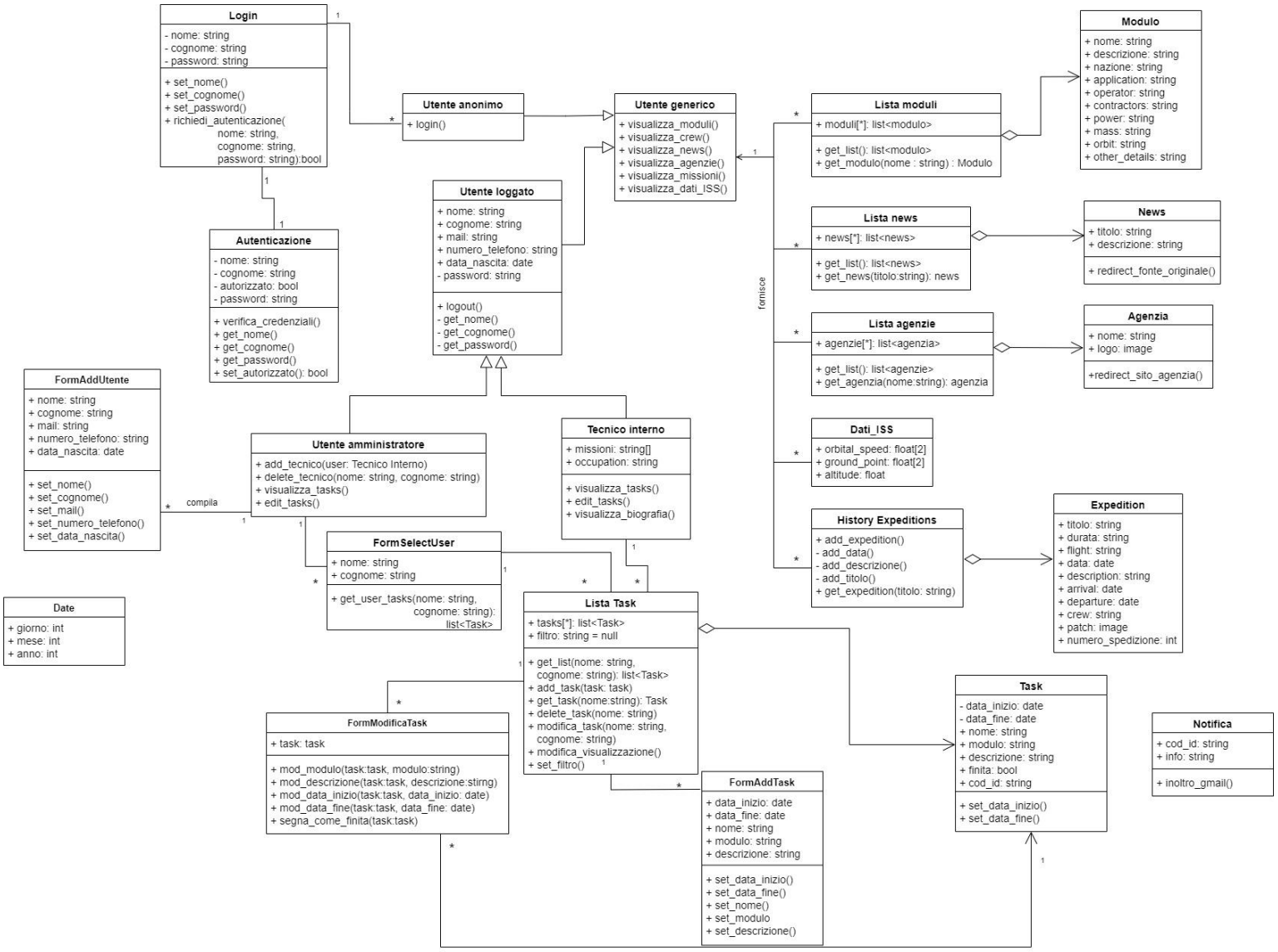


DIAGRAMMA DELLE CLASSI COMPLETO

Di seguito viene riportato il diagramma delle classi con tutte le classi fino ad ora presentate. Oltre alle classi già descritte, sono state inserite classi ausiliarie, ossia **Date** e **Notifica**. Queste classi servono per descrivere eventuali tipi strutturati usati, ad esempio, negli attributi delle altre classi.

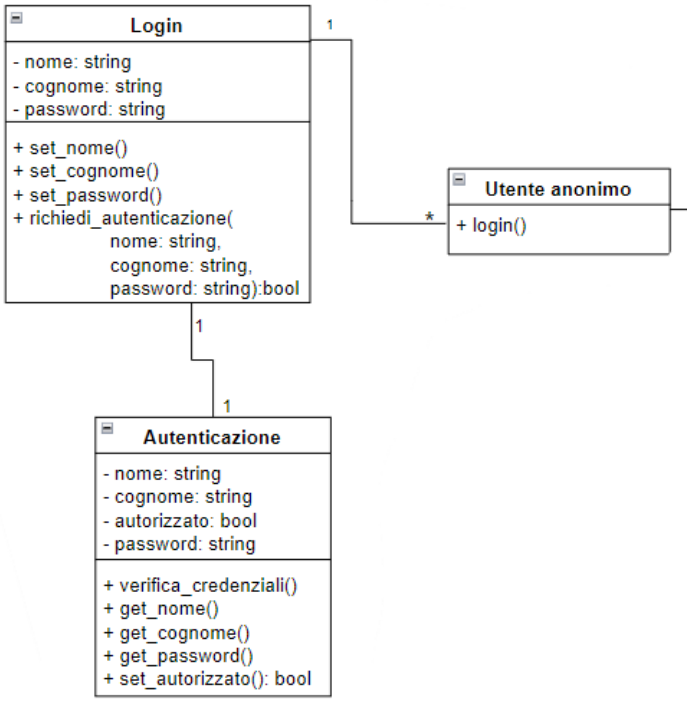


CODICE IN OBJECT CONSTRAINT LANGUAGE

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

AUTORIZZAZIONE LOGIN

Affinché l'operazione di login venga svolta con successo, vi sono le seguenti condizioni da rispettare, nel contesto delle classi Login e Autenticazione.



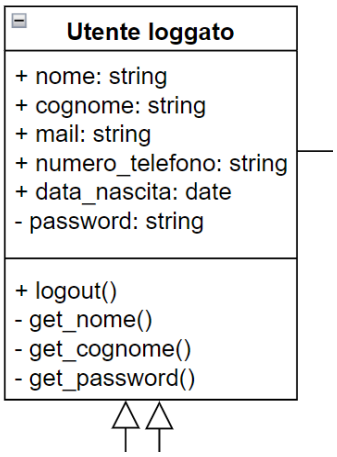
Prima di richiedere l'autenticazione bisogna compilare il nome e cognome e la password, e ciò viene espresso in OCL tramite una precondizione con questo codice:

```
context Login :: self.richiedi_autenticazione()
pre : self.nome != null and self.cognome != null and self.password != null
```

Inoltre, dopo che l'utente viene autorizzato, lo stato autorizzato viene messo a true, e ciò viene espresso in OCL tramite una postcondizione con questo codice:

context Autenticazione :: set_autorizzato()
post : self.autorizzato= true

UTENTE LOGGATO



Nel contesto della classe Utente Loggato, il numero di telefono e attributo di tipo stringa lungo 10 caratteri e ciò viene espresso in OCL tramite un'invariante con questo codice:

context Utente loggato
inv : lenght(self.numero_telefono)=10

GESTIONE delle TASK

Sono state identificate diverse condizioni che verificano la validità di certe operazioni sulle task, quali la completazione, l'aggiunta e l'eliminazione di una task e altre. Tutto ciò fa riferimento alle classi riportate in GESTIONE TASK (sezione 2.5 del documento). Tali condizioni vengono riportate qui in seguito.

Dopo che la task viene segnata come completata l'attributo finita viene settato su true, e ciò viene espresso in OCL tramite una postcondizione con questo codice:

context Task:: FormModificaTask.segna_come_finita(task:Task)
post : self.finita= true

Ovviamente la data inizio precede la data di fine, e ciò viene espresso in OCL tramite una invariante con questo codice:

context Task
inv : self.data_inizio < self.data_fine

Dopo che il filtro viene settato, l'attributo filtro assume un valore a seconda del filtro utilizzato, e ciò viene espresso in OCL tramite una postcondizione con questo codice:

```
context Lista Tasks :: set_filtro()
post : self.filtro != null
```

Le tasks dell'utente selezionato vengono restituite se sono presenti nella lista tasks, e ciò viene espresso in OCL tramite una precondizione con questo codice:

```
context FormSelectUser :: get_user_tasks(nome: string, cognome: string)
pre : ListaTasks.tasks -> includes(ListaTasks.get_list(nome: string, cognome: string))
```

Dopo l'aggiunta di una task essa deve essere presente nella lista, e ciò viene espresso in OCL tramite una postcondizione con questo codice:

```
context Lista Tasks :: add_task(task:Task)
post self.tasks -> includes(task: Task)
```

Dopo l'eliminazione di una task essa non deve essere più presente nella lista, e ciò viene espresso in OCL tramite una postcondizione con questo codice:

```
context Lista Tasks :: delete_task(nome: string)
post self.tasks -> excludes(get_task(nome: string))
```

Infine, la classe ausiliaria Date deve rispettare queste condizioni per essere valida, e ciò viene espresso in OCL tramite una invariante con questo codice:

```
context Date
inv : giorno > 0 and giorno <= 31 and mese > 0 and mese <= 12
```

DIAGRAMMA DELLE CLASSI E CODICE IN OCL

Qui viene riportato il diagramma delle classi con tutte le classi presentate insieme al codice OCL individuato.